# Towards Improving the Mental Model of Software Developers through Cartographic Visualization

Adrian Kuhn
Software Composition Group
University of Bern, Switzerland
http://scg.unibe.ch/akuhn

David Erni
Software Composition Group
University of Bern, Switzerland
http://www.deif.ch

Oscar Nierstrasz
Software Composition Group
University of Bern, Switzerland
http://scg.unibe.ch/oscar



## ABSTRACT

Software is intangible and knowledge about software systems is typically tacit. The mental model of software developers is thus an important factor in software engineering.

It is our vision that developers should be able to refer to code as being "up in the north", "over in the west", or "down-under in the south". We want to provide developers, and everyone else involved in software development, with a *shared*, *spatial* and *stable* mental model of their software project. We aim to reinforce this by embedding a cartographic visualization in the IDE (Integrated Development Environment). The visualization is always visible in the bottom-left, similar to the GPS navigation device for car drivers. For each development task, related information is displayed on the map. In this paper we present CODEMAP, an eclipse plug-in, and report on preliminary results from an ongoing user study with professional developers and students.

## Keywords

Architecture visualization, Human Factors, Mental Model, Spatial Representation, Software Development, Software Visualization, Tool Building.

## 1. INTRODUCTION

> *"If you write software knowledge down, it becomes immediately stale because it keeps changing in the developer's mind."* – Andrew Ko

Software is intangible and knowledge about software systems and their architecture is often tacit. The mental model of software developers is thus an important factor in software engineering. In our work, we aim to provide developers with tool support to establish a better mental model of their work. We do so by embedding a cartographic visualization, called CODEMAP, in their IDE (Integrated Development Environment). We do not aim to document or visualize the present mental model of developers, rather it is our goal that developers arrive at a better mental model based on the spatial visualization provided by our tool. This is motivated by the observation that the representation of source code in the IDE often impacts the mental model of developers. Compare for example the mental model held by an Eclipse developer with that of an `emacs` or `vim` user, or with the even more diverging mental model of development in explorative runtime systems such as Smalltalk and Self [14].

DeLine observed that developers are consistently lost in source code and that using textual landmarks only places a large burden on cognitive memory [5]. DeLine concluded that we need new visualization techniques that allow developers to use their spatial memory while navigating source code. He proposed four desiderata that should be satisfied by spatial software navigation [4]. In our most recent work [7] we generalized and extended this list as follows:

1. The visualization should show the entire program and be continuous.
2. The visualization should contain visualization landmarks that allow the developers to find parts of the system perceptually, rather than relying on naming or other cognitive feats.
3. The visualization should remain visually stable as the system evolves (both locally and across distributed version control commits).
4. The visualization should be capable of showing global information overlays.
5. Distance in the visualization should have a intuitive though technically meaningful interpretation.

We implemented the CODEMAP tool as a proof-of-concept prototype of Software Cartography. The prototype is open-source and available as an Eclipse plug-in[1]. At the moment, we are evaluating our approach in an ongoing controlled experiment with professional developers and students.

The remainder of this paper is structured as follows: Section 2 enumerates the developments tasks that are supported by the CODEMAP plug-in. Section 3 presents the codemap algorithm and its recent improvements. Section 4 discusses preliminary results from the ongoing evaluation. Section 5 discusses related work. Section 6 concludes with remarks on future work.

[1] `http://scg.unibe.ch/codemap`


*ICSE 2010,* Cape Town, South Africa

## 2. SOFTWARE CARTOGRAPHY

Software Cartography uses a spatial visualization of software systems to provide software development teams with a stable and shared mental model. Our cartographic visualization is most useful when it supports as many development tasks as possible. Therefore we integrated Software Cartography in the IDE so that a map of the software system may always be present and may thus support as many development tasks as possible.

At the moment, the CODEMAP plug-in for Eclipse supports the following tasks:[2]

- Navigation within a software system, be it for development or analysis. CODEMAP is integrated with the package explorer and editor of Eclipse. The selection in the package explorer and the selection on the map are linked. Open files are marked with an icon on the map. Double clicking on the map opens the closest file in the editor. When using heat map mode, recently visited classes are highlighted on the map.

- Comparing software metrics to each other, *e.g.*, to compre bug density with code coverage. CODEMAP is hooked into several Eclipse plug-ins in order to display their results on the map alongside the regular views. The map displays search results, compiler errors, and (given the ECLEMMA plug-in is installed) test coverage information. More information can be added through an extension point.

- Social awareness of collaboration in the development team. CODEMAP can connect two or more Eclipse instances to show open files of other developers. Colored icons are used to show the currently open files of all developers. Icons are colored by user and updated in real time.

- Help to understand a software system's domain. The layout of CODEMAP is based on structure *and* vocabulary, since we believe that a mental model of software should transcend structural artifacts. Labels on the map are not limited to class names, but include automatically retrieved keywords and topics.

- Exploring a system during reverse engineering. CODEMAP is integrated with Eclipse's structural navigation functions such as search for callers, implementers, and references. Arrows are shown for search results. We apply the FLOW MAP algorithm [12] to avoid visual clutter by merging parallel arrow edges. Figure 1 shows the result of searching for calls to the `#getSettingOrDefault` method in the `MenuAction` class .

## 3. THE CODEMAP ALGORITHM

Figure 2 illustrates the construction of a software map. The sequence of the construction is basically the same as presented in previous work [8, 7]. However, preliminary results from an ongoing user study have already led to a series of improvements which are discussed below.

**2-Dimensional Embedding:** A metric distance is used to compute the pair-wise dissimilarity of software artifacts

[2]New features are added on a weekly base, please subscribe to `http://twitter.com/codemap` to receive latest news.

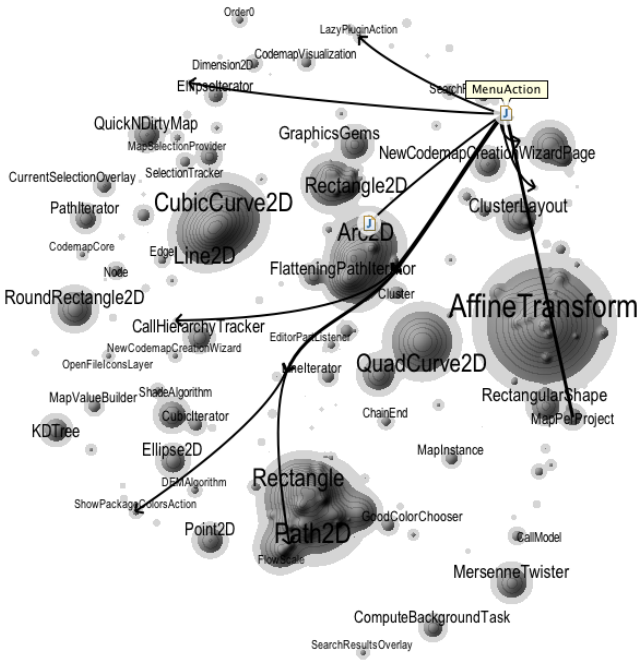


**Figure 1:** ***Thematic codemap of a software system, here the* Codemap *tool itself is shown. Arrow edges show incoming calls to the* `#getSettingOrDefault` ***method in the*** `MenuAction` ***class, which is currently active in the editor and thus labeled with a pop-up.***

(typically source code files). A combination of the ISOMAP algorithm [16] and MULTIDIMENSIONAL SCALING [1] is used to embed all software artifacts on the visualization pane[3]. The application of ISOMAP is an improvement over previous work in order to assist MDS with the global layout. The final embedding minimizes the error between the dissimilarity values and the visual distances.

Early prototypes of CODEMAP used a distance metric that was based on lexical similarity only. However, our user study revealed that developers tend to interpret visual distance as a measure of structural dependencies, even though they were aware of the underlying lexical implementation. Based on this observation, we developed an improved distance metric that takes both lexical similarity and structural distance (based on the "Law of Demeter" [11]) into account.

**Digital Elevation Model:** In the next step, a digital elevation model is created. Each software artifact contributes a Gaussian shaped basis function to the elevation model according to its KLOC size. The contributions of all software artifacts are summed up and normalized.

Using KLOC leads to an elevation model where large classes dominate the codemap. Observations from our user study indicate that this might be misleading since developers tend to interpret size as a measure of impact. Based on this observation, we implemented a set of different impact metrics [10] and plan to evaluate them in a fresh user study.

**Cartographic rendering:** In the final step, hill-shading

[3]Different to previous work LATENT SEMANTIC INDEXING is not applied anymore; it has been found to have little impact on the final embedding, if at all.

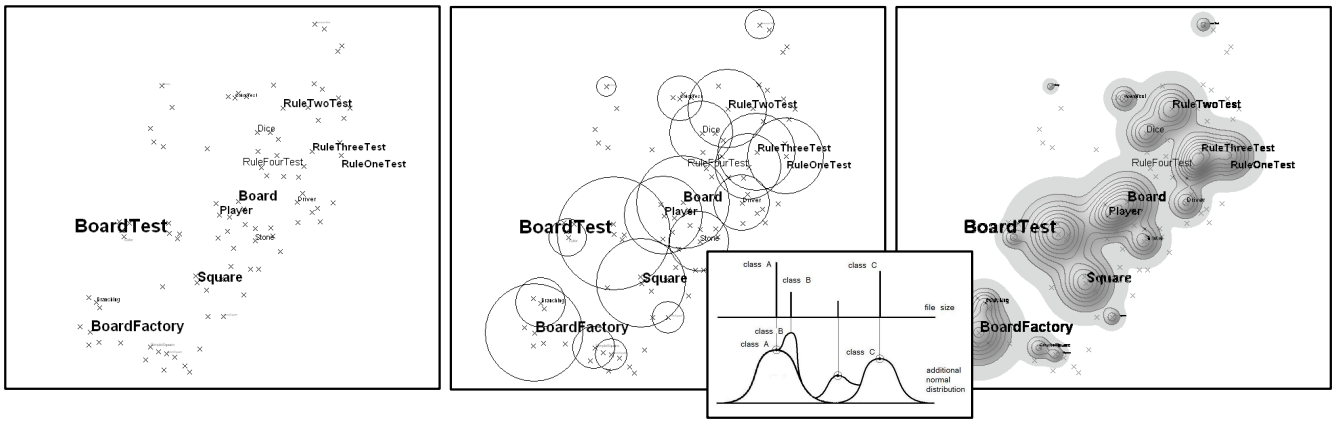


**Figure 2:** ***Construction steps of a software map, from left to right: 1) 2-dimensional embedding of files on the visualization pane; 2.a) circles around each file's location, based on class size in KLOC; 2.b) each file contributes a Gaussian shaped basis function to the elevation model according to its KLOC size; the contributions of all files are summed up; 3) fully rendered map with hill-shading, contour lines, and filename labels.***

is used to render the landscape of the software map. Please refer to previous work for full details [8]. Metrics and markers are rendered in transparent layers on top of the landscape. Users can toggle separate layer on/off and thus customize the codemap display to their needs.

## 4. PRELIMINARY RESULTS

At the moment, we are evaluating our approach in an ongoing controlled experiment with professional developers and students. The scenario of the experiment is first contact with an unknown closed-source system. Participants have 90 minutes to solve 5 exploratory tasks and to fix one bug report. After the experiment, participants are asked to sketch a drawing of their mental map of the system.

Preliminary results are mixed and challenge our assumptions on how developers would use the CODEMAP tool. Most importantly it became quickly apparent that we should revise our initial assumption that lexical similarity is a valid dissimilarity metric for the cartographic layout. Sofar, all participants tend to interpret visual distance as a measure of structural dependencies—even though they were aware of the underlying lexical implementation!

Developers intuitively expect that the map meets their mental model of the system's architecture. Since this was not given, developers were not able to take advantage of the map's consistent layout. So *e.g.*, even though north/south and east/west directions had clear (semantic) interpretations, developers did not navigate along these axes. Based on this observation we started to work on *anchored multi-dimensional scaling* such that developers may initialize the map to their mental model. Once a map has been initialized to a developer's mental model, the map is more likely to start co-evolving with and influencing the developer's mental model. Anchored MDS allows the developer to define anchors which influence the layout of the map [2]. Any software artifact can be used as an anchor, even those not present on the map as for example external libraries. With this future layout algorithm, developers may *e.g.*, arrange the database layer in the south and all UI layer in the north using the respective libraries as anchors.

Another observation was that inexperienced developers (*i.e.*, students) are more likely to find the map useful than professional developers. That was not unexpected, since to power users *any* new way of using the IDE is likely to slow them down, and conversely to beginners *any* way of using the IDE is novel. The only exception to this observation was CODEMAP's search bar, a one-click interface to Eclipse's native search, that was used by all participants but one.

In general, participants reported that CODEMAP was most useful when it displayed search results, callers, implementers, and references. A participant reported: *"I found it very helpful that you get a visual clue of quantity and distribution of your search results"*. In fact, we observed that that participants almost never used the map for direct navigation but often for search and reverse engineering tasks.

## 5. RELATED WORK

Most closely related to the work in this paper is the work on spatial representations of code[4] by the HIP (Human Interfaces in Programming) group of Microsoft Research.

DeLine proposed four desiderata that should be satisfied by spatial software navigation [4]. In the same work *Software terrain maps* are presented, which satisfy the properties #1 and #4 (*i.e.*, continuous space and global overlays).

Venolia and Cherubini ran a series of surveys and interviews on why and how developers use visual depictions of their code, *e.g.*, [3]. They found that diagrams that document design decisions were often externalized in temporary drawings and then subsequently lost. Most of the diagrams had a transient nature because of the high cost of changing whiteboard sketches to electronic renderings.

Code Canvas by Rowan [13], also with the HIP group, is a zoomable UML view that drops levels of details as you zoom out, and reveals code editors as you zoom in.

Using 2-dimensional embedding to visualize information based on the metaphor of cartographic maps is by no means a novel idea. *Topic maps*, as they are called, have a long-

[4]`http://research.microsoft.com/en-us/projects/spatialcode/`

standing tradition in information visualization. In fact, the work in this paper was originally inspired by media reports on Hermann and Leuthold's work on the political landscapes of Switzerland [6]. ThemeScape is the best-known example of a visualization tool that uses the metaphor of cartographic maps. Topics extracted from text documents are organized into a visualization based on topical distance and surface height corresponds to topical frequency [18].

In the software visualization literature however, topic maps are rarely used. Except for the use of graph splatting in RE Toolkit by Telea *et al.* [15], we are unaware of their prior application in software visualization.

A number of software visualization tools have adopted metaphors from cartography. Typically these tools are part of a~~n~~ reverse-engineering approach based on extracted models that abstract away from source code. Thus, these tools cannot be used to read source code or develop software. The two most comprehensive of these tools are:

CodeCity is an explorative environment based on the city metaphor [17]. CodeCity employs the nesting level of packages for their city's elevation model, and uses a modified tree layout to position packages and classes. Order is based on size, so the layout is not stable over time. CodeCity is not integrated into an IDE, but built in top of the Moose reverse-engineering platform that offers post-mortem analysis of abstract models only.

VERSO is an explorative environment that is also based on the city metaphor [9], very similar to CodeCity. VERSO employs a treemap layout to position their elements, which provides a more stable layout. However, VERSO is also limited to post-mortem analysis of abstract models only.

# 6. REMARKS ON FUTURE WORK

This paper presents Software Cartography, a spatial representation of software. Our approach is supposed to help developers with a better mental model of their software systems that is stable over time and shared with team mates. Preliminary results from an ongoing user study led to the revision of our assumption that lexical similarity is sufficient to layout the cartographic map [8]. We refined our layout algorithm based on that conclusion. The new layout is based on both lexical similarity and the ideal structural proximity proposed by the "Law of Demeter".

As future work, we can identify the following promising directions:

- Software maps at present are largely static. We envision a more interactive environment in which the user can "zoom and pan" through the landscape to see features in closer detail, or navigate to other views of the software.
- Selectively displaying features would make the environment more attractive for navigation. Instead of generating all the labels and thematic widgets up-front, users can annotate the map, adding comments and waymarks as they perform their tasks.
- Orientation and layout are presently consistent for a single project only. We would like to investigate the usefulness of conventions for establishing consistent layout and orientation (*i.e.*, "testing" is North-East) that will work across multiple projects, possibly within a reasonably well-defined domain.

**Acknowledgments:** We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Bringing Models Closer to Code" (SNF Pro ject No. 200020-121594, Oct. 2008 – Sept. 2010).

# 7. REFERENCES

[1] I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[2] A. Buja, D. F. Swayne, M. L. Littman, N. Dean, H. Hofmann, and L. Chen. Data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, 17(2):444–472, June 2008.

[3] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: how and why software developers use drawings. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 557–566, New York, NY, USA, 2007. ACM.

[4] R. DeLine. Staying oriented with software terrain maps. In *DMS*, pages 309–314, 2005.

[5] R. DeLine, A. Khella, M. Czerwinski, and G. G. Robertson. Towards understanding programs through wear-based filtering. In *SOFTVIS*, pages 183–192, 2005.

[6] M. Hermann and H. Leuthold. *Atlas der politischen Landschaften*. vdf Hochschlverlag AG, ETH Zürich, 2003.

[7] A. Kuhn, D. Erni, P. Loretan, and O. Nierstrasz. Software cartography: Thematic software visualization with consistent layout. *Journal of Software Maintenance and Evolution (JSME)*. To appear.

[8] A. Kuhn, P. Loretan, and O. Nierstrasz. Consistent layout for thematic software maps. In *Proceedings of 15th Working Conference on Reverse Engineering (WCRE'08)*, pages 209–218, Oct. 2008.

[9] G. Langelier, H. Sahraoui, and P. Poulin. Visualization-based analysis of quality for large-scale software systems. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 214–223, 2005.

[10] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

[11] K. J. Lieberherr, I. M. Holland, and A. Riel. Object-oriented programming: An objective sense of style. In *Proceedings OOPSLA '88, ACM SIGPLAN Notices*, volume 23, pages 323–334, Nov. 1988.

[12] D. Phan, L. Xiao, R. Yeh, and P. Hanrahan. Flow map layout. In *IEEE Symposium on Information Visualization (INFOVIS '05)*, pages 219–224, October 2005.

[13] K. Rowan. Code canvas. http://www.webcitation.org/5mceC6NVX, March 2009. `http://blogs.msdn.com/kaelr/archive/2009/03/26/code-canvas.aspx`, archived at `http://www.webcitation.org/5mceC6NVX`.

[14] D. W. Sandberg. Smalltalk and exploratory programming. *SIGPLAN Not.*, 23(10):85–92, October 1988.

[15] A. Telea, A. Maccari, and C. Riva. An open toolkit for prototyping reverse engineering visualizations. In *Symposium on Data Visualisation 2002 (VISSYM '02)*, pages 241–ff. Eurographics Association, 2002.

[16] J. B. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[17] R. Wettel and M. Lanza. Visualizing software systems as cities. In *Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis)*, pages 92–99, 2007.

[18] J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow. Visualizing the non-visual: spatial analysis and interaction with information from text documents. *infovis*, 00:51, 1995.